

Engineering of Advanced Software Solutions (EASS)

HIT, Israel

Yossi Eliaz

2022

- Admin
- Technical debt
- Business logic
- Bash and commandline (based on MIT's missing semester)
- Git, GitHub
- Interactive class



- Discord
- Github account
- HW on Github
- Creating a Canvas account
(<https://canvas.instructure.com/>)
- Accepted invitation from AWS Academy and GitHub (I have sent links)
- AWS on Cavnvas
- LinkedIn
- Commandline (WSL)
- Docker
- Moodle (minimal interaction over there)
- Volunteer to summarize the lectures
- Stackoverflow
- Engagment on Discord
- Hackernews

References:

Missing Semester MIT
AWS cloud certificate

What is Technical Debt?

- “In software development, there is always a constant need to balance speed and quality. Some quality will always have to be sacrificed to release features within a reasonable timeframe, so any of these shortcuts will often be tasked as future projects. Those unattended tasks become what is called technical debt.”
- “There are several reasons why technical debt happens. Product owners may focus more on the need to implement and release new features and less on fixing past problems or create a generic enough infrastructure to support future developments. In some severe cases, product owners completely underestimate the outcomes of dealing with poor infrastructure, bugs and poorly designed software.”
- “Ultimately, technical debt can sometimes lead to software users having bad experiences and thereby increasing user churn rates. Together, a lack of developer awareness and task ownership can lead to more technical debt.”

References:

<https://logz.io/blog/technical-debt/>

- “**Business rules** are what your non-software developers tell you what your software needs to do.”
- “**Business logic** is the part of your code that specifically implements business rules.”



References:

<https://softwareengineering.stackexchange.com/questions/234251/what-really-is-the-business-logic>

<http://www.ritholtz.com>

- Vim
- Bash
- Git
- Docker

Important tools and commands

- echo, while, find, vars, printenv, htop, shebang, wild cards
- cp, touch, mkdir, ls, uniq, awk, rm
- man man
- brew
- wget
- curl

References:

<https://missing.csail.mit.edu/2020/shell-tools/>

- `docker run`
- `docker ps`
- `docker run -ti`

Must have:

- 1 Problem statement
- 2 Sample code and data
- 3 Spelling, grammar and formatting

Example:

<https://stackoverflow.com/questions/11227809/why-is-processing-a-sorted-array-faster-than-processing-an-unsorted-array>

References:

<https://codeblog.jonskeet.uk/2010/08/29/writing-the-perfect-question/>
<https://stackoverflow.com/help/how-to-ask>

- Checkout github classroom and the first task about git and github
<https://classroom.github.com/classrooms/99552739-eass-hit-2022-part-a>

- S3, EC2, RDS, and EBS modules
- must get 100 on all 4 modules
- grading will be 25% per module

A bit more about EC2 instances and types of hardwares (HW)

EC2 provides secure, resizable compute cloud services. It makes web-scale cloud computing easier and offers HW such as:

- ARM vs. Intel vs. AMD (x86, x86_64)
- GPUs (Nvidia, Intel)
- TPUs (on Google Computing Platform)
- Metal instances on AWS
- FPGA-based nodes

Instance Types and prices (useful links)

<https://aws.amazon.com/ec2/instance-types/>
<https://instances.vantage.sh/>

Instance Types (summary)

- 1 General Purpose
- 2 Compute Optimised
- 3 Memory Optimised
- 4 Accelerated Computing (P instances are for general-purpose GPU applications)

Pricing

There are four ways to pay for EC2 instances: On-Demand, Reserved Instances, and Spot Instances & Per-Second Billing. You can also pay for Dedicated Hosts which provide you with EC2 instance capacity on physical servers dedicated for your use.

Checkout github classroom and the first task about git and github

<https://classroom.github.com/classrooms/99552739-eass-hit-2022-part-a>

How to test our code/system

General approaches for testing

- Static vs. Dynamic
- Passive testing
- White-box vs. Black-box testing

Types of testing coverage metric

- API testing – testing all public and private APIs
- Code coverage – creating tests to satisfy some criteria of code coverage (e.g., the test designer can create tests to cause all statements in the program to be executed at least once)

Types of testing systems (CI/CD)

- Unit vs. Integration testing
- System testing
- Compatibility testing
- Installation testing
- Smoke and sanity testing
- Regression testing

We will use pytest and fastapi testing system

- <https://fastapi.tiangolo.com/tutorial/testing/>
- <https://docs.pytest.org/>

- 1 4 modules on AWS course (S3, EC2, EBS, RDS) - if you finish all the course you get +10 bonus points to final grade
- 2 Build full REST/HTTP fastapi backend + Dockerization (due 1/4)
- 3 UI (react/streamlit) (due 1/5)
- 4 Docker compose the server with UI and backend plus server and write a clear README with git submodules (due 29/5)
- 5 Presentation of the system in a demo in a 2-3 minutes video on youtube and clear README (due 29/5)

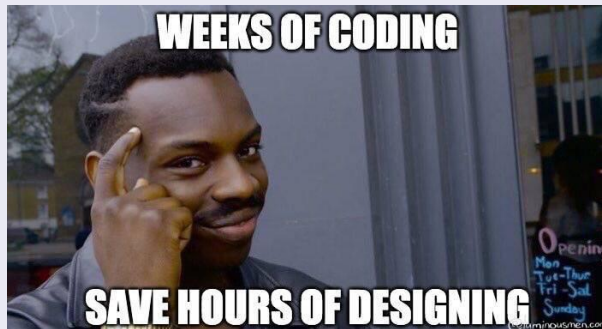
Ideas for projects next semester (based on skills we will learn this semester)

- 1 AI/ML based predictive system
- 2 Smart contracts (web3)
- 3 Any other system with at least 3 microservices

- Monolithic vs. Microservices
- Docker
- Client-Server
- REST/HTTP API
- FastAPI
- Pytest
- asyncio
- Frontend (React javascript and Streamlit python)
- Docker compose
- Functional programming
- How to compile a new library

What Are The Best Software Engineering Principles?

Measure twice and cut once



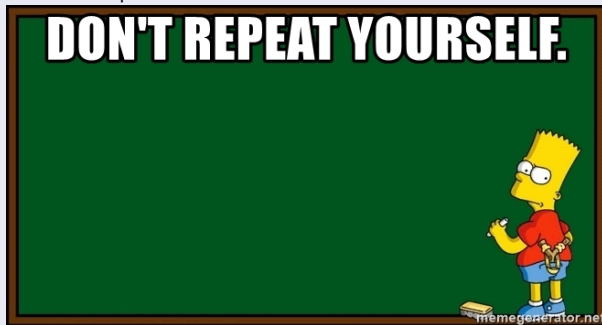
Based on this nice post

<https://luminousmen.com/post/what-are-the-best-engineering-principles>

What Are The Best Software Engineering Principles?

Don't Repeat Yourself (DRY)

If any code occurs more than twice in the codebase, you should think of moving it in a separate function. In fact, you should consider creating a separate method even if you encounter repetition a second time.



What Are The Best Software Engineering Principles?

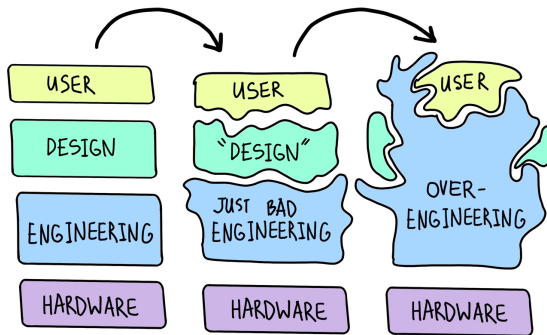
Keep It Simple -Stupid- (KISS)

Some think that this idea transformed from Occam's Razor philosophical principle. You can interpret it as follows: one should not create extra entities to the system without a strong necessity. It is always a good idea to first consider the usefulness of adding another method/class/tool/process, etc.

What Are The Best Software Engineering Principles?

You Aren't Gonna Need It (YAGNI)

Don't implement all the "necessary" (most likely unnecessary) functionality at once from the very beginning of the project.



@luminousmen.com

What Are The Best Software Engineering Principles?

Avoid Premature Optimization

“Premature optimization is the root of all evil (or at least most of it) in programming” — Donald Knuth

Watch Knuth on a talk with Lex Friedman <https://www.youtube.com/watch?v=EE1R8FYUJm0>

Principle Of Least Astonishment

This principle means that your code should be intuitive and obvious, and not surprise another developer when reviewing the code.

Law of Demeter (Olympian goddess of the harvest and agriculture)

The basic idea here is to divide the areas of responsibility between classes and encapsulate the logic within a class, method, or structure.

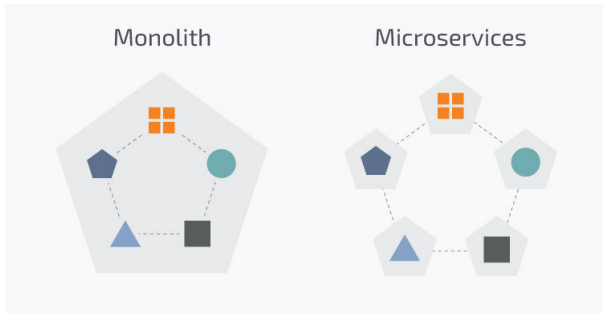
- 1 **Decoupling** You should try to reduce the number of connections between different classes or entities
- 2 **Cohesion** The associated classes must be in one module/package/directory

SOLID - create code that is easy to maintain and extend over time

- Single responsibility states that every module or class should have responsibility for a single part of the functionality and that responsibility should be entirely encapsulated by the class
- Open-closed states that software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification
- Liskov substitution states that any inherited class should complement (substitutable), not replace, the behavior of the base class
- Interface segregation states that no client of the class should be forced to depend on methods it does not use
- Dependency inversion says that programmers should work at the interface level and not at the implementation level

Monolithic vs. Microservices

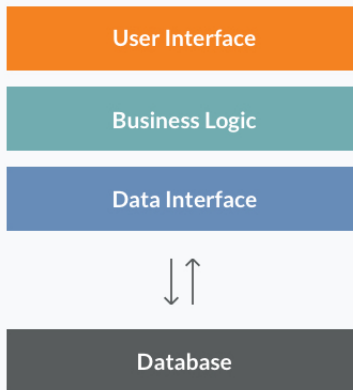
- Monolithic application is a single unified unit that contains all the logic in one entity
- Microservice architecture breaks the application down into a collection of smaller independent units



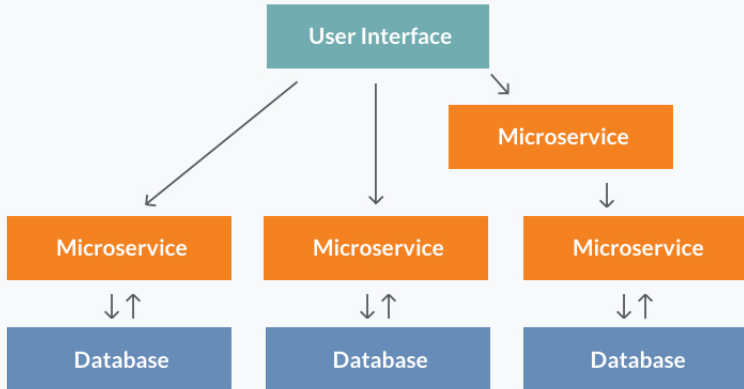
Further reading material

<https://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business>

Monolithic Architecture



Microservice Architecture



- Dockerhub/Registry
- Dockerfile
- `docker build`
- `docker run`
- `docker ps`
- `docker network ls`
- `docker volumes`
- `docker expose ports`
- `docker images`
- `docker exec`
- `docker image prune -a`

Further training material

<https://training.play-with-docker.com/alacart/>

<https://towardsdatascience.com/twenty-one-techniques-and-five-concepts-for-better-docker-usage-9ee135dccc9>

<https://www.docker.com/blog/how-to-use-the-official-nginx-docker-image/>

- ❶ `docker run -it --rm -d -p 8080:80 --name web nginx`
- ❷ `curl http://localhost:8080`
- ❸ `docker stop web`
- ❹ Add `index.html` to local `site-content` and map it to `/usr/share/nginx/html`
(<https://gist.github.com/chrisvfritz/bc010e6ed25b802da7eb>)
- ❺ `docker run -it --rm -d -p 8080:80 --name web -v`
`~/site-content:/usr/share/nginx/html nginx`
- ❻ Doing stuff via Dockerfile (`docker build -t webserver`):
`FROM nginx:latest`
`COPY ./index.html /usr/share/nginx/html/index.html`
- ❼ `docker run -it --rm -d -p 8080:80 --name web webserver`

“A very common scenario for developers, is to run their REST APIs behind a reverse proxy. There are many reasons why you would want to do this but one of the main reasons is to run your API server on a different network or IP then your front-end application is on. You can then secure this network and only allow traffic from the reverse proxy server. For the sake of simplicity and space, I've created a simple frontend application in React.js and a simple backend API written in Node.js. Run the following command to pull the code from GitHub.”

Go through (reverse proxy, react, nginx):

<https://www.docker.com/blog/how-to-use-the-official-nginx-docker-image/>



Be active on EASS discord and try to learn and help each other as much as you can.

<https://docs.microsoft.com/en-us/visualstudio/docker/tutorials/docker-tutorial>
<https://github.com/docker/awesome-compose/tree/master/fastapi>
<https://luminousmen.com/post/what-are-the-best-engineering-principles>